1. *Your project report must be TYPE-WRITTEN using* 12*-point font. It should NOT be more than* 4 *pages (inclusive of diagrams).*

1 The Game of Tetris

Tetris is a popular video game played on a two-dimensional grid, which we assume to be of size 20 rows by 10 columns in this project. Each square in the grid can be full or empty, making up a "wall of bricks" with "holes". The squares fill up as objects of different shapes fall from the top of the grid and are added to the top of the wall, giving rise to a "jagged top". Each falling object can be moved horizontally and can be rotated by the player in all possible ways, subject to the constraints imposed by the sides of the grid. There is a finite set of standard shapes for the falling objects, which we assume to comprise 7 possible object shapes in this project as shown in Figure 1. The game starts with an empty grid and ends when a square in the top row becomes full and the top of the wall reaches the top of the grid. However, when a row of full squares is created, this row is removed, the bricks lying above this row move one row downward, and the player scores 1 point. More than one row can be created and removed in a single step, in which case the number of points scored by the player is equal to the number of rows removed. The player's objective is to maximize the score attained (total number of rows removed) up to termination of the game.

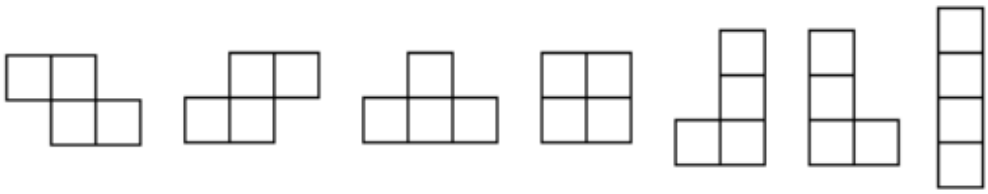

Figure 1: The 7 Tetris object shapes, each of which is an arrangement of 4 connected squares.

2 Heuristics for Designing a Fast, Proficient Tetris Player

The goal of this project is to develop a competent utility-based agent to play the Tetris game. The agent's action, denoted by a, is the horizontal positioning and rotation applied to the falling object. Each state of the game consists of two components:

1. The board position, that is, a binary description of the full/empty status of each square, denoted by i.

2. The shape of the current falling object, denoted by y.

As soon as the most recent object has been placed, the new component y is generated according to a probability distribution $p(y)$, independently of the preceding history of the game. In this project, we assume $p(y)$ to be a uniform distribution.

Unfortunately, the number of states in the Tetris problem is extremely large. It is roughly equal to $m2^{hw}$, where m is the number of different shapes of falling objects, and h and w are the height and width of the grid, respectively. In particular, this project assumes $m = 7$, $h = 20$, and $w = 10$, which produce over 10^{61} states. This motivates the need to approximate the utility function of the Tetris player with, for example, a good heuristic/evaluation function. In particular, we can construct a heuristic function that evaluates a Tetris board position based on some characteristic features of the position. Such features are easily recognizable by experienced players, and include the current height of the wall, the presence of "holes" and "glitches" (severe irregularities) in the first few rows, etc. Consequently, the agent's utility function can be approximated by a heuristic function that comprises a linear weighted sum of features[1]

$$\widetilde{V}(i) = \omega(0) + \sum_{k=1}^{n} \omega(k)\phi_k(i)$$

where n is the number of features, $\omega = (\omega(0), \omega(1), \ldots, \omega(n))$ is the weight vector, and $\phi_k(i), k = 1, \ldots, n$, are the feature functions for board position i. For example, the feature functions can be defined to be

[1] It is a common approach to specify a heuristic function using a linear weighted sum of features. For example, see sections 3.6.4 and 5.4.1 of AIMA 3rd edition.

1. $\phi_1(i), \ldots, \phi_{10}(i)$ correspond to the 10 column heights of the wall;
2. $\phi_{11}(i), \ldots, \phi_{19}(i)$ are the absolute difference between adjacent column heights;
3. $\phi_{20}(i)$ is the maximum column height; and
4. $\phi_{21}(i)$ is the number of holes in the wall, that is, the number of empty positions of the wall that are surrounded by full positions.

Note that you do not have to adhere to the feature functions that are specified above. You are allowed to specify your own feature functions. Even though you may be tempted to handcode your Tetris agent by simply specifying the values of the weight vector ω, this project requires you to learn ω using data (e.g., see ref2.pdf).

Given board position i and shape of current falling object y, the agent's strategy μ for playing the Tetris game is defined as follows:

$$\mu(i, y) \triangleq \arg\max_a \; r(i, y, a) + \widetilde{V}(f(i, y, a))$$

where $r(i, y, a)$ and $f(i, y, a)$ are, respectively, the reward (i.e., number of rows removed) and the resulting new board position when action a is applied to the falling object y in board position i.

3 Criteria

Project report: This 4-page report should document your agent's strategy, experimental results to demonstrate its performance, and observations and analysis to discuss why your strategy performs well (or not). State clearly and explicitly the *novel* and *significant* contributions and achievements (if any) of your team that, in your opinion, will distinguish your agent's strategy and project work from that of the other teams and the existing literature. In particular, thanks to the recent hype on big data, you need to demonstrate that your agent's *learning* method can scale up to big data, that is, it has to be capable of learning a competent/proficient Tetris agent using millions, billions, or even trillions of Tetris objects in a sufficiently short time. Describe, in the report, your proposed novel learning method that is *EXPLICITLY*[3] designed and implemented to handle big data and empirically demonstrate through experiments that it can indeed scale up to big data (e.g., by reporting the speedup[4]) while preserving the agent's good learning performance. One possibility is to consider parallel/distributed learning on multiple cores/machines[5].
Wow! factor: Basically, when I read your report, I should go like "Wow! This learning method is really cool!" In a technically sound and pleasantly surprising way, of course. Then, you'll get the marks in this category.
Performance against the agents designed by other project teams: We will test your agent against our favorite sequence of object shapes, which is of course not revealed to you! Be ready to demo your program when called upon by the TAs. You will not receive any points in this category if your code cannot be compiled on our machine.
Project program: A template written in java is provided. For the java code, you should only be modifying the "PlayerSkeleton.java" file[6]. You should provide detailed comments in your code to facilitate our understanding. Zip your completed "PlayerSkeleton.java" code with the project report for submission. You will not receive any points in this category if your code cannot be compiled on our machine.
Following instructions: The instructions are provided on page 1.